Systemy RT i embedded

Wykład 9
Interfejsy mikrokontrolerów,
cz. I

Wrocław 2013
Plan

- Microncontrollers’ interfaces
- SCI
- SPI
- I²C
- OneWire
- I²S
Microcontrollers’ interfaces
Types of interfaces

Interface:

a. *equipment or programs designed to communicate information from one system of computing devices or programs to another.*

b. *any arrangement for such communication.*
Types of interfaces

• Types:
  – Software
  – Hardware

  – Serial
  – Parallel:
    • 4-bit
    • 8-bit
    • 16-bit
    • ...
Serial vs parallel

• Features of serial interfaces:

  – Simplicity:
    • Medium (RS232)
    • Large (OneWire)

  – Performance
    • From very small (OneWire) to very large (Ethernet)

  – Implementation difficulties:
    • From small (RS232) to large (Bluetooth)
Serial vs parallel

• Features of parallel interfaces:
  – Simplicity:
    • Medium
  – Performance
    • Large or very large (64-bit)
    • Limited by propagation delay, noises
  – Implementation difficulties:
    • Small at low speed
    • Very large at high speeds
    • Problems with large range implementations
UART interface
UART interface

• Features:
  – One of the most popular serial interfaces
  – Serial, two-wire interface
  – Used for internal and external communication
  – Original versions (e.g. RS-232C) require many control lines (DB9, DB25)
  – Two state logic used, but not TTL!
  – Asynchronous data transfer. **No clock** transmitted!
  – Full duplex transmission
  – Point – to – point transmission
  – Throughput up to hundreds of kb/s
### RS232C

![Diagram of RS232C connector](attachment:image)

<table>
<thead>
<tr>
<th>Numer</th>
<th>Kierunek</th>
<th>Oznaczenie</th>
<th>Nazwa angielska</th>
<th>Nazwa polska</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 pin 25 pin</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>DCE -&gt; DTE</td>
<td>DCD</td>
<td>sygnał wykrycia nośnej</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>DCE -&gt; DTE</td>
<td>RxD</td>
<td>odbiór danych</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>DCE &lt; DTE</td>
<td>TxD</td>
<td>transmisja danych</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>DCE &lt; DTE</td>
<td>DTR</td>
<td>gotowość terminala 1)</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>DCE - DTE</td>
<td>GND</td>
<td>masa</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>DCE -&gt; DTE</td>
<td>DSR</td>
<td>gotowość &quot;modemu&quot; 1)</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>DCE &lt; DTE</td>
<td>RTS</td>
<td>żądanie wysyłania</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>DCE -&gt; DTE</td>
<td>CTS</td>
<td>gotowość wysyłania</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>DCE -&gt; DTE</td>
<td>RING</td>
<td>wskaźnik dzwonka</td>
</tr>
<tr>
<td>9-19; 21; 23-25</td>
<td></td>
<td>NC</td>
<td></td>
<td>nie wykorzystane 2)</td>
</tr>
</tbody>
</table>
USART in STM32F4

• Features:
  - Full duplex, asynchronous or **synchronous** communications
  - Fractional baud rate generator systems - Common programmable transmit and receive baud rate
  - Programmable data word length (8 or 9 bits)
  - Configurable stop bits - support for 1 or 2 stop bits
  - Transmitter clock output for synchronous transmission
  - Single-wire half-duplex communication
  - Configurable multibuffer communication using DMA (direct memory access)
    • Buffering of received/transmitted bytes in reserved SRAM using centralized DMA
USART in STM32F4

9-bit word length (M bit is set), 1 stop bit

Data frame

Possible parity bit

Next data frame

Start bit

9-bit word length (M bit is set), 1 stop bit

Idle frame

Break frame

8-bit word length (M bit is reset), 1 stop bit

Data frame

Possible parity bit

Next data frame

Start bit

Idle frame

Break frame

** LBCL bit controls last data clock pulse
USART in STM32F4

- For asynchronous mode only two pins necessary:
  - **RX**: Receive Data Input
  - **TX**: Transmit Data Output

- Additional pin necessary for synchronous mode:
  - **SCLK**: Transmitter clock output. This pin outputs the transmitter data clock for synchronous transmission

- Pins required in Hardware flow control mode:
  - **nCTS**: Clear To Send blocks the data transmission at the end of the current transfer when high
  - **nRTS**: Request to send indicates that the USART is ready to receive a data (when low).
USART in STM32F4

- Important registers:
  - USART_SR – status register
  - USART_DR – data register (8-bit value)
### USART in STM32F4

**USART_BRR – baud rate register**

- **Address offset:** 0x08
- **Reset value:** 0x0000 0000

#### Table

<table>
<thead>
<tr>
<th>S.No</th>
<th>Desired</th>
<th>Actual</th>
<th>Value programmed in the baud rate register</th>
<th>% Error (=) (Calculated - Desired B. rate) / Desired B. rate</th>
<th><strong>f(_{PCLK}) = 8 MHz</strong></th>
<th><strong>f(_{PCLK}) = 12 MHz</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>38.4 Kbps</td>
<td>38.462 Kbps</td>
<td>13</td>
<td>0.16</td>
<td>38.339 Kbps</td>
<td>10.5625</td>
</tr>
<tr>
<td>6</td>
<td>57.6 Kbps</td>
<td>57.554 Kbps</td>
<td>8.6875</td>
<td>0.08</td>
<td>57.692 Kbps</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>115.2 Kbps</td>
<td>115.942 Kbps</td>
<td>4.3125</td>
<td>0.64</td>
<td>115.385 Kbps</td>
<td>6.5</td>
</tr>
<tr>
<td>8</td>
<td>230.4 Kbps</td>
<td>228.571 Kbps</td>
<td>2.1875</td>
<td>0.79</td>
<td>230.769 Kbps</td>
<td>3.25</td>
</tr>
<tr>
<td>9</td>
<td>460.8 Kbps</td>
<td>470.588 Kbps</td>
<td>1.0625</td>
<td>2.12</td>
<td>461.538 Kbps</td>
<td>1.625</td>
</tr>
<tr>
<td>10</td>
<td>921.6 Kbps</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>11</td>
<td>2 MBps</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>12</td>
<td>3 MBps</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
USART in STM32F4

• Important registers:
  – USART_CR1 – control register
  – USART_CR2 – control register
  – USART_CR3 – control register
USART in STM32F4

- TC/TXE behaviour when transmitting
USART in STM32F4

- Parity control:
  - Parity control - generation of parity bit in transmission and parity checking in reception
  - Can be enabled by setting the PCE bit in the USART_CR1 register
  - Parity can be Even or Odd
USART in STM32F4

• Synchronous mode:
  - Full duplex, clock controlled mode used for fast data transfer to slave devices
**USART in STM32F4**

- Synchronous mode:

![USART Synchronous Mode Diagram](image-url)
USART in STM32F4

- **Half-duplex operation:**
  - the TX and RX lines are internally connected
  - the RX pin is no longer used
  - the TX pin is always released when no data is transmitted. Thus, it acts as a standard I/O in idle or in reception.
USAR in STM32F4

- Continuous operation with DMA:
  - Transmission

![Diagram showing USART operation with DMA](image-url)
UART in STM32F4

- Continuous operation with DMA:
  - Reception
USART in STM32F4

- Hardware flow control:
  - It is possible to control the serial data flow between 2 devices by using the nCTS input and the nRTS output
USART in STM32F4

• Hardware flow control:
  - Request To Send - RTS flow control
  - nRTS is asserted (tied low) as long as the USART receiver is ready to receive a new data
USART in STM32F4

- Hardware flow control:
  - Clear To Send - CTS flow control

- the transmitter checks the nCTS input before transmitting the next frame
USART in STM32F4

- Multiprocessor communication:
  - There is a possibility of performing multiprocessor communication with the USART (several USARTs connected in a network)
  - One of the USARTs is a master - its TX output is connected to the RX input of the other USARTs
  - The outputs of slaves are ANDed and connected to the RX of the master
  - Each processor has each own address (soft support)
USART in STM32F4

• USART Interrupts

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit Data Register Empty</td>
<td>TXE</td>
<td>TXEIE</td>
</tr>
<tr>
<td>CTS flag</td>
<td>CTS</td>
<td>CTSIE</td>
</tr>
<tr>
<td>Transmission Complete</td>
<td>TC</td>
<td>TCIE</td>
</tr>
<tr>
<td>Received Data Ready to be Read</td>
<td>RXNE</td>
<td>RXNEIE</td>
</tr>
<tr>
<td>Overrun Error Detected</td>
<td>ORE</td>
<td></td>
</tr>
<tr>
<td>Idle Line Detected</td>
<td>IDLE</td>
<td>IDLEIE</td>
</tr>
<tr>
<td>Parity Error</td>
<td>PE</td>
<td>PEIE</td>
</tr>
<tr>
<td>Break Flag</td>
<td>LBD</td>
<td>LBDIE</td>
</tr>
<tr>
<td>Noise Flag, Overrun error and Framing Error in multibuffer communication</td>
<td>NF or ORE or FE</td>
<td>EIE</td>
</tr>
</tbody>
</table>
SPI Interface
SPI interface

• Features:
  – One of the most popular serial interfaces
  – Serial, four-wire interface
  – Used for internal communication
  – Master-slave architecture
  – Two state TTL logic used!
  – Fully synchronous data transfer. Clock controlled by the master!
  – Full duplex transmission
  – Point – to – point transmission
  – Throughput up to tens of Mb/s
- **MISO** - *master input slave output*
- **MOSI** - *master output slave input*
- **SCLK** - *serial clock*
- **SS** - *slave select*
- Both in Master and in Slave there are implemented shift registers (8-bits), SIPO and PISO type
- Zata are exchanged with the SCK signal
- After 8 (16) clock cycles the data transfer is finished
SPI - configuration
SPI - Multiple devices

• **Data bus**

• **Daisy chain**
SPI in STM32F4

- Main features:
  - Full-duplex synchronous transfers on three lines
  - Simplex synchronous transfers on two lines with or without a bidirectional data line
  - 8- or 16-bit transfer frame format selection
  - Master or slave operation
  - Multimaster mode capability
  - Programmable clock polarity and phase
  - Programmable data order with MSB-first or LSB-first shifting
  - Hardware CRC feature for reliable communication
  - 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests
SPI in STM32F4
SPI in STM32F4

- Half-duplex operation
  - The SPI is capable of operating in half-duplex mode in 2 configurations:
    - 1 clock and 1 bidirectional data wire
    - 1 clock and 1 data wire (receive-only or transmit-only)
SPI in STM32F4

- Data transfer in full-duplex Master mode
SPI in STM32F4

- Data transfer in full-duplex Slave mode
SPI in STM32F4

• Main registers:
  - SPI_DR - 16b data register split into 2 buffers  
    - one for writing (Transmit Buffer) and another one for reading (Receive buffer)
  - SPI_SR - status register
SPI in STM32F4

- Main registers:
  - SPI_CR1 - control register 1
  - SPI_CR2 - control register 2
• SPI communication using DMA:
  - SPI can be configure to operate at throughput exceeding 10Mb/s
  - To facilitate the transfers, the SPI features a DMA capability implementing a simple request/acknowledge protocol
  - In transmission, a DMA request is issued each time TXE is set to 1. The DMA then writes to the SPI_DR register.
  - In reception, a DMA request is issued each time RXNE is set to 1. The DMA then reads the SPI_DR register
SPI in STM32F4

- Transmission using DMA
SPI in STM32F4

- Reception using DMA
SPI in STM32F4

- Interrupts:

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable Control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit buffer empty flag</td>
<td>TXE</td>
<td>TXEIE</td>
</tr>
<tr>
<td>Receive buffer not empty flag</td>
<td>RXNE</td>
<td>RXNEIE</td>
</tr>
<tr>
<td>Master Mode fault event</td>
<td>MODF</td>
<td></td>
</tr>
<tr>
<td>Overrun error</td>
<td>OVR</td>
<td>ERRIE</td>
</tr>
<tr>
<td>CRC error flag</td>
<td>CRCERR</td>
<td></td>
</tr>
<tr>
<td>TI frame format error</td>
<td>FRE</td>
<td>ERRIE</td>
</tr>
</tbody>
</table>
I²C Interface
I²C interface

• Features:
  – Serial, two-wire interface
  – Used for internal communication
  – Master-slave architecture
  – Two state TTL logic used
  – Fully synchronous data transfer. Clock controlled by the master
  – Half-duplex transmission
  – Point – to – point transmission
  – Data rate 400kb/s (3.4Mb/s in HS mode)
Features:
- Transmission over two lines
- Both lines are open-drain type
- Both lines are pulled-up
- Each device checks, if there is no collision on the line
- Data transfer can be initiated only by the master
I²C - START/STOP

I²C - data transfer
I²C - Acknowledge
I²C - addressing

- Each device has its own unique address
- Address is 7-bit long
- 8-th bit defines if there will be a read (1) or write (0) operation


I²C - special modes

• In the first protocol specifications (1982) the max speed was set to 100 kb/s
• Next the Fast Mode was introduced with the maximum speed of 400 kb/s
• In 2006 the Fast Mode Plus was defined with maximum speed of 1Mb/s
• With additional logic the, so called, Highspeed Mode can be implemented with maximum speed of 3.4 Mb/s
I²C in STM32F4

- Main features:
  - Multimaster capability: the same interface can act as Master or Slave
  - I²C Master features:
    - Clock generation
    - Start and Stop generation
  - I²C Slave features:
    - Programmable I²C Address detection
    - Dual Addressing Capability to acknowledge 2 slave addresses
    - Stop bit detection
I²C in STM32F4

- Main features:
  - Generation and detection of 7-bit/10-bit addressing
  - Supports different communication speeds:
    - Standard Speed (up to 100 kHz)
    - Fast Speed (up to 400 kHz)
  - Programmable digital noise filter
  - Optional clock stretching
  - 1-byte buffer with DMA capability
I²C in STM32F4

• Modes of operation:
  - Slave transmitter
  - Slave receiver
  - Master transmitter
  - Master receiver

  - By default the module operates in Slave mode
  - The interface automatically switches from slave to Master:
    • after it generates a START condition
  - The interface automatically switches from master to slave:
    • if an arbitration loss
    • a Stop generation occurs, allowing multimaster capability
I²C in STM32F4
I²C in STM32F4

- Slave transmitter

7-bit slave transmitter

- S Address A Data1 A Data2 A \ldots DataN NA P
  - EV1 EV3-1 EV3 EV3 EV3 \ldots EV3-2

10-bit slave transmitter

- S Header A Address A
  - EV1
- S Header A Data1 A \ldots DataN NA P
  - EV1 EV3_1 EV3 EV3 EV3 \ldots EV3-2

Legend: S= Start, S*= Repeated Start, P= Stop, A= Acknowledge, NA= Non-acknowledge, EVx= Event (with interrupt if ITEVFEN=1)

EV1: ADDR=1, cleared by reading SR1 followed by reading SR2
EV3-1: TxE=1, shift register empty, data register empty, write Data1 in DR.
EV3: TxE=1, shift register not empty, data register empty, cleared by writing DR
EV3-2: AF=1; AF is cleared by writing '0' in AF bit of SR1 register.
I²C in STM32F4

- Slave receiver

Legend: S = Start, S_r = Repeated Start, P = Stop, A = Acknowledge, EVx = Event (with interrupt if ITEVFEN=1)

EV1: ADDR=1, cleared by reading SR1 followed by reading SR2
EV2: RxNE=1 cleared by reading DR register.
EV4: STOPF=1, cleared by reading SR1 register followed by writing to the CR1 register
I²C in STM32F4

- Master transmitter

Legend: S= Start, Sr = Repeated Start, P= Stop, A= Acknowledge, EVx= Event (with interrupt if ITEVFEN = 1)

- **EV5**: SB=1, cleared by reading SR1 register followed by writing DR register with Address.
- **EV6**: ADDR=1, cleared by reading SR1 register followed by reading SR2.
- **EV8_1**: TxE=1, shift register empty, data register empty, write Data1 in DR.
- **EV8**: TxE=1, shift register not empty ,data register empty, cleared by writing DR register
- **EV8_2**: TxE=1, BTF = 1, Program Stop request. TxE and BTF are cleared by hardware by the Stop condition
- **EV9**: ADD10=1, cleared by reading SR1 register followed by writing DR register.
I²C in STM32F4

- Master receiver

Legend: S = Start, S_r = repeated Start, P = Stop, A = Acknowledge, NA = Non-acknowledge, EVx = Event (with interrupt if ITEVFEN=1)
EV5: SB=1, cleared by reading SR1 register followed by writing DR register.
EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2. In 10-bit master receiver mode, this sequence should be followed by writing CR2 with SART = 1.
In case of the reception of 1 byte, the Acknowledge disable must be performed during EV6 event, i.e. before clearing ADDR flag.
EV7: RxNE = 1 cleared by reading DR register.
EV7_1: RxNE = 1 cleared by reading DR register, programming ACK = 0 and STOP request.
EV9: ADD10 = 1, cleared by reading SR1 register followed by writing DR register.
I²C in STM32F4

- DMA:
  - DMA requests (when enabled) are generated only for data transfer
  - DMA requests are generated:
    - by Data Register becoming empty in transmission
    - Data Register becoming full in reception
### I²C in STM32F4

#### Interrupts:

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Event flag</th>
<th>Enable control bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start bit sent (Master)</td>
<td>SB</td>
<td>ITEVFEN</td>
</tr>
<tr>
<td>Address sent (Master) or Address matched (Slave)</td>
<td>ADDR</td>
<td>ITEVFEN</td>
</tr>
<tr>
<td>10-bit header sent (Master)</td>
<td>ADD10</td>
<td>ITEVFEN</td>
</tr>
<tr>
<td>Stop received (Slave)</td>
<td>STOPF</td>
<td>ITEVFEN</td>
</tr>
<tr>
<td>Data byte transfer finished</td>
<td>BTF</td>
<td>ITEVFEN and ITBUFEN</td>
</tr>
<tr>
<td>Receive buffer not empty</td>
<td>RxNE</td>
<td>ITEVFEN and ITBUFEN</td>
</tr>
<tr>
<td>Transmit buffer empty</td>
<td>TxE</td>
<td>ITEVFEN and ITBUFEN</td>
</tr>
<tr>
<td>Bus error</td>
<td>BERR</td>
<td>ITERREN</td>
</tr>
<tr>
<td>Arbitration loss (Master)</td>
<td>ARLO</td>
<td>ITERREN</td>
</tr>
<tr>
<td>Acknowledge failure</td>
<td>AF</td>
<td>ITERREN</td>
</tr>
<tr>
<td>Overrun/Underrun</td>
<td>OVR</td>
<td>ITERREN</td>
</tr>
<tr>
<td>PEC error</td>
<td>PECERR</td>
<td>ITERREN</td>
</tr>
<tr>
<td>Timeout/Tlow error</td>
<td>TIMEOUT</td>
<td>ITERREN</td>
</tr>
<tr>
<td>SMBus Alert</td>
<td>SMBALERT</td>
<td>ITERREN</td>
</tr>
</tbody>
</table>
I²C in STM32F4

• Main registers:
  - I2C_CCR - Clock control register
  - I2C_DR - data register
  - I2C_OAR1 - Own address register
  - I2C_SR1 - status register 1
  - I2C_SR2 - status register 2
  - I2C_CR1 - control register 1
  - I2C_CR2 - control register 2
OneWire Interface
OneWire interface

• Features:
  – Serial, one-wire interface
  – Communication wire can also be used for delivering power supply!
  – Used for internal communication but on longer distances
  – Master-slave architecture
  – Asynchronous data transfer. Data transfer controlled by the master
  – Half-duplex transmission
  – Point-to-point transmission
  – Data rate 15.4kb/s (standard) or 125kb/s (overdrive)
OneWire - podłączenie master/slave
OneWire - reset/wykrywanie obecności

- Precence detection
OneWire - zapis/odczyt

LEGEND
- PULLUP
- MASTER
- SLAVE
- SPEED = STANDARD (15.4kbps)

WRITE 1
WRITE 0
READ 1
READ 0

SLAVE DEVICE(S) SAMPLE LINE
SLAVE DEVICE(S) SAMPLE LINE

T = 0µs  T = 15µs  T = 60µs
15µs < T < 60µs

S₀…Sₙ OVERDRIVE RESISTOR
• Communication has three main phases: RESET, authorisation (ROM commands), data read/write
• Each OneWire device has a unique 64-bit security code
OneWire – zasilanie przez magistrale

- Power supply delivery
I²S Interface
I²S interface

• Features:
  – Serial, bidirectional bus used to transfer data in electronic devices
  – Suitable for connection of digital audio
  – Developed by Philips (like I2C)
  – Also known abbreviation IIS
  – Separated clock and data bus (similar to SPI)
  – Transmission is via at least three lines SCK (clock), WS (select dates), SD (data) and ground GND
  – Speed depends on the type of data being transferred
  – Maximum speed exceeds 2 Mb / s (eg 32b data at 44.1kHz)
I²S interface

- Features, cont’d:
  - On the I²S bus there is only one transmitter and one monitoring device (the master)
  - The master can be a transmitter, a receiver or a system of supervising transmission between two slave devices
  - I²S interface transfers data of two channels: left and right
  - Data channels are transmitted on change
  - A large number of devices used for the transmission of additional supervisory controller
I²S configurations
\( \text{I}^2\text{S} - \text{time dependencies (transmitter)} \)

### Example: Master transmitter with data rate of 2.5MHz (±10%)

<table>
<thead>
<tr>
<th></th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock period ( T )</td>
<td>360</td>
<td>400</td>
<td>440</td>
<td>( T_{tr} = 360 )</td>
</tr>
<tr>
<td>clock HIGH ( t_{HC} )</td>
<td>160</td>
<td></td>
<td></td>
<td>min &gt; 0.35 ( T ) = 140 (at typical data rate)</td>
</tr>
<tr>
<td>clock LOW ( t_{LC} )</td>
<td>160</td>
<td></td>
<td></td>
<td>min &gt; 0.35 ( T ) = 140 (at typical data rate)</td>
</tr>
<tr>
<td>delay ( t_{dr} )</td>
<td></td>
<td>300</td>
<td></td>
<td>max &lt; 0.80 ( T ) = 320 (at typical data rate)</td>
</tr>
<tr>
<td>hold time ( t_{hw} )</td>
<td>100</td>
<td></td>
<td></td>
<td>min &gt; 0</td>
</tr>
<tr>
<td>clock rise-time ( t_{RC} )</td>
<td></td>
<td>60</td>
<td></td>
<td>max &gt; 0.15 ( T_{tr} ) = 54 (only relevant in slave mode)</td>
</tr>
</tbody>
</table>
I²S - time dependencies (receiver)

\[ T = \text{clock period} \]
\[ T_r = \text{minimum allowed clock period for transmitter} \]
\[ T > T_r \]

Example: Slave receiver with data rate of 2.5MHz (±10%) (all values in ns)

<table>
<thead>
<tr>
<th></th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock period (T)</td>
<td>360</td>
<td>400</td>
<td>440</td>
<td>(T_r = 360)</td>
</tr>
<tr>
<td>clock HIGH (t_{HC})</td>
<td>110</td>
<td>126</td>
<td></td>
<td>(\text{min} &lt; 0.35T = 126)</td>
</tr>
<tr>
<td>clock LOW (t_{LC})</td>
<td>110</td>
<td>126</td>
<td></td>
<td>(\text{min} &lt; 0.35T = 126)</td>
</tr>
<tr>
<td>set-up time (t_{sr})</td>
<td>60</td>
<td></td>
<td></td>
<td>(\text{min} &lt; 0.20T = 72)</td>
</tr>
<tr>
<td>hold time (t_{sh})</td>
<td>0</td>
<td></td>
<td></td>
<td>(\text{min} &lt; 0)</td>
</tr>
</tbody>
</table>
I²S - transmitter diagram
I2S - receiver diagram
PCM3001 - Stereo Codec with analog output

- Low-cost system encoder / decoder audio
- Includes digital filters, provides digital signal attenuation, De-emphasis, mild blanking signal (soft-mute) and the detection of silence
- Control via a digital interface
PCM3001 - Block diagram
PCM3001 - I²S

FORMAT 5: FMT[2:0] = 101

DAC: 18-Bit, MSB-First, I²S

ADC: 18-Bit, MSB-First, I²S
Thank you for your attention
References